I CLAIM:

Sub B15 D1

1. For a computer-executable program that operates on a data structure, where the data structure must have a required state at selected program points, a method of transforming said program comprising the steps of:

(A) analyzing the program to determine the state of said data structure at said selected program points;

(B) partitioning said determined state at each said program point into components that may each be set separately;

(C) determining the operations required to set each component of the state at each selected program point; and

(D) placing said operations in a way that eliminates partial redundancies of said operations.

2. The method of claim 1, wherein the data structure stores items on a first-in last-out basis.

3. The method of claim 2, wherein the states of the data structure are represented as paths on a tree of nodes where:

(A) each path traverses the tree towards the root, and

(B) each node on the path represents a component of the state.

4. The method of claim 2, wherein the data structure represents actions to be taken by the program if an exceptional situation arises.

5. The method of claim 4, wherein the selected program points are the points of execution immediately before instructions that might cause an exceptional situation.

6. The method of claim 5, wherein the actions to be taken are represented explicitly as exceptional paths in a graph before the transformation, and said exceptional paths are removed.

7. A method of eliminating partial redundancy, which includes the steps of computing down-safety and up-safety, comprising the steps of:

(A) computing down-safety speculatively by ignoring rarely taken branches in the control-flow graph; and

(B) computing up-safety using the results of the down-safety calculation to determine where operations are speculatively available.

8. The method of claim 7, wherein the computation of down-safety further comprises a lattice that distinguishes strict down-safety from speculative down-safety.

A

9. For a computer-executable program that keeps track of actions to take in event of an exceptional situation by maintaining a stack, a method of inserting instructions to maintain said stack comprising the steps of:

(A) representing the program with a control-flow graph in which actions to take in event of an exceptional situation are represented by explicit paths in the graph;

(B) analyzing said program to determine at which points the stack must be in a valid state;

(C) building a forest of trees that represent the stack state at said points where:

(i) each node of the tree represents a possible item on the stack, and

(ii) a stack state is represented as a path from a tree node to the root of the tree;

(D) computing where to place operations that set the state of items on the stack by performing the steps of:

(i) constructing flow equations for down-safety of operations that set the state of items on the stack, where the equations ignore rarely taken branches,

(ii) solving said flow equations for down-safety of operations that set the state of items on the stack,

(iii) constructing flow equations for up-safety of operations that set the state of items on the stack, where the equations use the solution for down-safety to determine which setting operations are speculatively available, and

(iv) solving said flow equations for up-safety of operations that set the state of items on the stack,

(E) using the results of said flow equations to place instructions that maintain the stack;

(F) removing edges from the control-flow graph which represent said actions for exceptional events; and

(G) inserting a prologue at entry to the control-flow graph that saves the existing pointer to the top of the EH stack.

Add $B^2$7